

# Chapter 8

## Instruction Cache

This chapter describes the structure and function of the Instruction Cache. The Instruction Cache acts as a buffer memory between external memory and the DSP core processor. When code executes, the code words at the locations requested by the instruction set are copied into the Instruction Cache for direct access by the core processor. If the same code is used frequently in a set of program instructions, storage of these instructions in the cache yields an increase in throughput because external bus accesses are eliminated. In the DSP56300 instruction set are specific cache instructions that permit you to lock sectors of the cache and to flush the cache contents under software control. The Instruction Cache controls 1K of Instruction Cache memory, with the following features:

- Software-controlled Cache Enable (CE) bit in the Extended Mode Register (EMR) in the Status Register (SR)<sup>1</sup>
- Eight-way, fully associative Instruction Cache with sectored placement policy
- 1- to 4-word transfer granularity
- Least Recently Used (LRU) sector replacement algorithm
- Transparent operation (that is, no user management is required)
- Individual sector locking/unlocking
- Global cache flush controlled by software
- Cache controller status observable via the JTAG/OnCE port

**Note:** Supported Instruction Cache size is device-dependent. Refer to the device-specific technical data sheet to determine the Instruction Cache size for a device.

---

1. For details on the Status Register (SR), see **Section 5.4.1.2**, "Status Register (SR)."

## 8.1 Instruction Cache Architecture

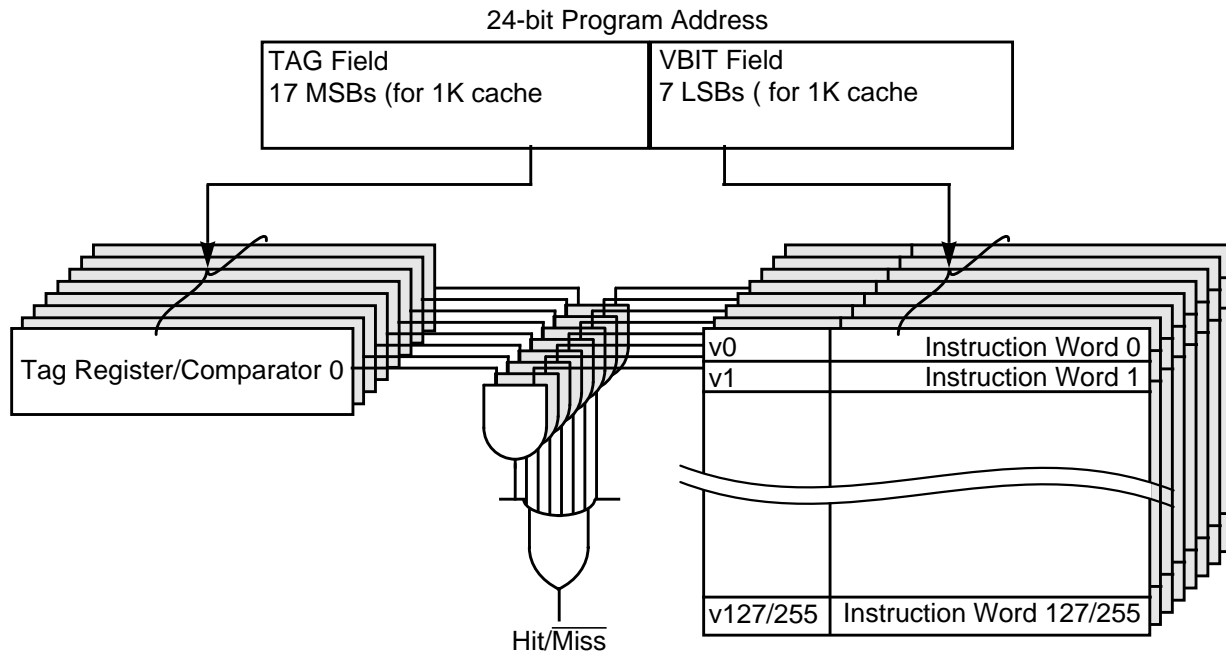
The Instruction Cache is composed of the following:

- **Memory Array:** The actual memory space defined for use by the Cache Controller is 1024 24-bit words and is logically divided into eight 128-word cache sectors. The sector placement algorithm is fully associative. Each word has an associated source address to identify the cache contents. Since the Cache Controller treats Program RAM as 128-word sectors, the 24-bit address is divided into the following two fields:
  - VBIT field: 7 LSBs (for 1K cache) for the word displacement in the sector
  - TAG field: 17 MSBs (for 1K cache) for the sector base address
- **Tag Register File:** Contains the TAG fields of the base addresses of the memory sectors currently mapped into the cache.
- **Valid Bit Array:** Contains a set of valid bits for each possible address in a referenced memory sector. There are valid bits arranged as eight banks of 128 bits each, one bank for every sector. A bit is set if the address location is already in the cache. If the bit is cleared, an external memory fetch is required. Notice that you cannot directly access these valid bits. Processor hardware reset clears the valid bits to indicate that the Program RAM content is not initialized.
- **Cache Controller:** When the Program Control Unit (PCU) initiates a program fetch request, the Cache Controller compares the TAG field of the requested address to tags in each of the eight Memory Array sectors. All eight sectors are searched in parallel using the eight comparators in the Cache Controller. Then the Cache Controller determines whether the request is a cache hit or miss. For cache hits, the address contents are transferred as directed by the PCU for execution. For cache misses, the Cache Controller initiates a fetch in coordination with the Sector Replacement Unit.
- **Sector Replacement Unit (SRU):** When a sector miss occurs<sup>1</sup>, the SRU determines which sector is flushed from the cache by monitoring requested addresses and sector usage and replacing the least recently used (LRU) sector. The LRU stack status is affected by instruction fetch operations and PFLUSH, PLOCK, and PUNLOCK program cache instructions. Locked cache sectors continue to move up and down the LRU stack, but when the LRU sector is picked, locked sectors are skipped. When initialized by reset, the LRU stack default is from sector number 0 (Most Recently Used) to sector number 7 (LRU).

---

1. If there is no match between the tag field and all sector tag registers, meaning that the memory sector containing the requested word is not present in the cache, the situation is called a *sector miss*. A sector miss is another form of a cache miss.

**Figure 8-1** shows a block diagram of the Instruction Cache.



**Figure 8-1.** Instruction Cache Block Diagram

## 8.2 Cache Programming Model

The Instruction Cache is controlled by two control bits:

- Cache Enable (CE) bit in the Extended Mode Register (EMR) part of the Status Register (SR Bit 19)

When CE is cleared, the Instruction Cache is disabled. When CE is set, the Instruction Cache is enabled.

- Burst Enable (BE) bit in the Extended Operating Mode (EOM) part of the Operating Mode Register (OMR Bit 10)

When BE is cleared, the Instruction Cache transfer on a miss is one word. When BE is set, the Instruction Cache transfer on a miss increases to a burst block of one to four words.

**Note:** To ensure proper operation, do not clear the Cache Enable mode (CE bit in SR) while Burst mode is enabled (BE bit in OMR is set). Refer to **Chapter 5, Program Control Unit** for details on the SR and OMR.

- The instruction set supports the Instruction Cache via the following instructions:
  - PLOCK
  - PLOCKR
  - PUNLOCK
  - PUNLOCKR
  - PFREE
  - PFLUSH
  - PFLUSHUN

## 8.2.1 Cache Operation

When enabled, the cache is involved in every instruction fetch. Its actions depend on several conditions, including whether the program address is (cache hit) or is not (cache miss) in the Instruction Cache and whether Burst mode is enabled or disabled. The following paragraphs describe the conditions under which the Instruction Cache operates.

### 8.2.1.1 Program Fetch

When the core generates an address for an instruction fetch, the cache controller compares its TAG field to the tag values currently stored in the Tag Register File.

### 8.2.1.2 Cache Hit

If a tag match (that is, sector hit) exists, then the valid bit of the corresponding word in that cache sector is checked using the VBIT field as an address to the Valid Bit Array. If the valid bit is set, meaning the word in the cache is valid, then that word is fetched from the cache location corresponding to the desired address. This situation is called a cache hit, meaning that both corresponding sector and corresponding instruction word are present and valid in the Instruction Cache. The Sector Replacement Unit (SRU) flags the sector as the Most Recently Used (MRU).

### 8.2.1.3 Cache Word Miss When Burst Mode Is Disabled

If a tag match (that is, sector hit) exists, and Burst Mode is disabled, but the desired word is not flagged as valid (corresponding valid bit is cleared), then the cache initiates a read access to the external program memory, introducing wait states into the pipeline. The number of wait states is the number of wait states programmed into the Bus Control registers (BCRs) plus one, reflecting the type of memory used. The Sector Replacement Unit (SRU) flags the sector as the Most Recently Used (MRU), and the fetched instruction is sent to the core and copied to the relevant sector location. Then the valid bit of that word is set.

### 8.2.1.4 Cache Word Miss When Burst Mode Is Enabled

If a tag match (that is, sector hit) exists, and Burst Mode is enabled, but the desired word is not flagged as valid (that is, the corresponding valid bit is cleared), then the cache initiates a burst of up to four read accesses to the external program memory. The exact number of fetch requests depends on the value of the two LSBs of the address of the initiating fetch that was detected as a miss, as indicated in **Table 8-1**.

**Table 8-1.** Determining the Number of Required Fetches in Burst Mode

Value of the 2 LSBs of the Requested Address	Number of Fetch Requests Initiated
00	Four requests are initiated
01	Three requests are initiated
10	Two requests are initiated
11	Only one request is initiated(that is, same as if the Burst mode is disabled)

These external read accesses introduce wait states into the pipeline. The number of wait states for each fetch is the number of wait states that are programmed into the bus control registers (BCRs) plus one, reflecting the type of memory used. The Sector Replacement Unit (SRU) flags the sector as the Most Recently Used (MRU), and each of the fetched instructions is copied to the relevant sector location. Then the valid bit of that word is set.

### 8.2.1.5 Sector Miss

If there is no match between the TAG field and all sector Tag registers, meaning that the memory sector containing the requested word is not in the cache, the situation is called a sector miss, which is another form of a cache miss. If a sector miss occurs, the SRU selects the sector to be replaced. The cache controller then flushes the selected cache sector by clearing all corresponding valid bits, loads the corresponding Tag register with the new TAG field, and simultaneously initiates an access to the external Program RAM, as described in **Section 8.2.1.3** and **Section 8.2.1.4**. The sector is flagged as MRU, the fetched instruction is sent to the core and copied to the relevant sector location, and the valid bit of that word is set.

## 8.2.2 Default Mode After Hardware Reset

After hardware reset, the Instruction Cache is disabled. The cache is initialized as follows:

- All valid bits are cleared.
- All Tag Registers are initialized to ‘all ones,’ that is, \$1FFFF for a 1K Cache (17-bit Tag Register).
- The LRU stack holds a default descending order of sectors (from seven to zero).
- All cache sectors are in the unlocked state.

## 8.3 Cache Locking

Cache locking is useful for locking some time-critical code parts in the cache memory. When a cache sector is locked, the Sector Replacement Unit (SRU) cannot replace this sector, even if it becomes the Least Recently Used (LRU) sector (bottom of LRU stack). A sector can be locked by the instructions PLOCK or PLOCKR. The operand for these instructions is an effective memory address (absolute or program counter-relative). The cache sector to which this address belongs, if one exists, is locked. If the specified effective address does not belong to one of the current cache sectors, a memory sector containing this address is allocated into the cache, thereby replacing the LRU cache sector. This cache sector is locked, but empty. If all the cache sectors are already locked, this memory sector is not allocated into the cache, and the lock operation is not executed. The locked cache sector becomes MRU. Locking a cache sector already in the cache does not affect its contents, the value of its valid bits, or the corresponding Tag Register contents.

**Note:** PLOCK and PLOCKR are detected as illegal opcodes when the Instruction Cache is not enabled. Issuing these instructions when the cache is disabled initiates the Illegal Interrupt. A distance of at least 3 instruction cycles (equivalent to three NOP instructions) should be maintained between an instruction that changes the value of the Cache Enable bit (CE) and one of the instructions PLOCK and PLOCKR.

## 8.4 Cache Unlocking

A locked sector can be unlocked to allow sector replacement from that cache sector. Unlocking can be performed in three different ways.

- A locked sector is unlocked by the PFREE, PUNLOCK, or PUNLOCKR instructions. The operands of the PUNLOCK and PUNLOCKR instructions are effective memory addresses (absolute or program counter-relative). The memory sector containing this address is allocated into a cache sector, if it is not already in a

cache sector, and this cache sector is unlocked. If all the cache sectors are already locked, this memory sector is not allocated into the cache, and the unlock operation is not executed. The unlocked cache sector becomes MRU and is enabled for replacement by the LRU algorithm. Unlocking a locked cache sector using these instructions does not affect its contents, its tag, or its valid bits.

- All locked sectors are unlocked simultaneously using the instruction PFREE, which allows you to reset the locking mechanism. Unlocking the sectors using PFREE neither affects the sector contents (instructions already fetched into the sector storage area), valid bits, tags, nor the LRU stack status.
- The locked sectors are unlocked by the PFLUSH instruction. Unlocking the sectors via PFLUSH clears all the sectors' valid bits and sets the LRU stack and Tag registers to their default values.

**Note:** PFREE, PUNLOCK and PUNLOCKR are detected as illegal opcodes when the Instruction Cache is not enabled. Issuing these instructions when the cache is disabled initiates the Illegal Interrupt. A distance of at least three instruction cycles (equivalent to three NOP instructions) should be maintained between an instruction that changes the value of the Cache Enable bit (CE) and one of the instructions PFREE, PUNLOCK and PUNLOCKR.

## 8.5 Flushing the Cache

Executing the PFLUSH or PFLUSHUN instructions flushes the cache. Executing PFLUSH causes a global cache flush that brings the cache to the following hardware reset initial condition:

- All valid bits are cleared.
- All Tag Registers are initialized to 'all ones,' that is, \$1FFFF for a 1K Cache (17-bit Tag Register).
- The LRU stack holds a default descending order of sectors (from 7 to 0).
- All cache sectors are in the unlocked state.

Executing PFLUSHUN causes a flush only to the unlocked sectors and initializes the cache as follows:

- All valid bits of the unlocked sectors are cleared.
- All Tag Registers of the unlocked sectors are initialized to 'all ones,' that is, \$1FFFF for a 1K Cache (17-bit Tag Register).
- The LRU stack holds a default descending order of sectors (from 7 to 0).

**Note:** Coherency between Program RAM mode and Cache mode is not supported by the Instruction Cache Controller. It is not possible to fill the cache while in Program RAM mode and use the contents after switching to Cache mode. The cache is automatically flushed when switching from Cache to Program RAM mode.

**Note:** PFLUSH and PFLUSHUN are detected as illegal opcodes when the Instruction Cache is not enabled. Issuing these instructions when the cache is disabled initiates the Illegal Interrupt. At least three instruction cycles (equivalent to three NOP instructions) should be maintained between an instruction that changes the value of the Cache Enable bit (CE) and one of the instructions PFLUSH and PFLUSHUN.

## 8.6 Data Transfers to/from Instruction Cache

Data transfers to/from the program memory can be accomplished by the DMA or by software, using MOVE instructions. Only PMOVE instructions can transfer data to/from the Instruction Cache.

### 8.6.1 DMA Transfers

DMA transfers have no effect on the Tag Register File, Valid Bit Array and LRU Stack, even when the cache is enabled. When the cache is disabled, the Instruction Cache memory space is considered part of the internal program memory space. DMA transfers to/from this space execute without any limitation. When the cache is enabled, the Instruction Cache memory space is considered part of the external program memory space. DMA transfers to/from this space execute through the external memory expansion port. Coherency between the external program memory and the contents of the Instruction Cache is not maintained.

### 8.6.2 Software-Controlled Transfers

The term “PMOVE” indicates use of a MOVE instruction to transfer data between the program memory space and any other source/destination. PMOVE data transfers do not affect the Tag Register File and LRU Stack, even if the cache is enabled. The term “PMOVEW” indicates a PMOVE transfer with the program memory space as the destination. The term “PMOVER” indicates a PMOVE transfer with the program memory space as the source.

When the cache is disabled, the Instruction Cache memory space is considered part of the internal program memory space. PMOVER from this space or PMOVEW to this space

execute without any limitation. When the cache is enabled, the cache controller checks the PMOVE transfers for a hit or miss:

- If the cache controller generates a hit on the program memory space address, the data is read from the cache memory array. Since PMOVE is not considered an instruction fetch operation, the LRU state is not changed by this transfer.
- If the cache controller generates a miss on the program memory space address, the data is read from the external program memory. The Cache state is not changed by this transfer. In Burst mode, no burst is initiated. Be aware that the core is delayed by the number of wait states specified in the BCR.

When the cache is enabled, the cache controller checks the PMOVEW transfers for a hit or miss:

- If the cache controller generates a sector hit on the program memory space address, the data is written both to the cache memory array and to the external program memory. The valid bit of the word is set. The LRU stack is not changed by this transfer. Be aware that the core is delayed by the number of wait states specified in the BCR.
- If the cache controller generates a sector miss on the program memory space address, the data is written only to the external program memory. The Cache state is not changed by this transfer. In Burst mode, no burst is initiated. Be aware that the core is delayed by the number of wait states specified in the BCR.

**Note:** For proper operation, none of the three instructions before a PMOVE transfer should clear or set the Status Register CE bit.

## 8.7 Using the Instruction Cache in Real-Time Applications

The following tips help you to use the Instruction Cache in real-time applications:

- Each sector (out of the 8, 128 words) can be individually locked.
- Locking a sector prevents its replacement in case of a miss even if it would have been its turn to be replaced.
- It is typical to lock the interrupt vector tables and routines to ensure the fastest response. Furthermore, these routines can be loaded beforehand using PMOVEs to ensure a hit on the first access.
- The cache can be globally flushed (for example, for task switching) with one instruction.
- The cache can be globally unlocked (that is any sector can be replaced in case of a miss) or any individual sector can be unlocked allowing its replacement.

- The penalty incurred for a cache miss is identical with the one for a regular instruction fetch from external memory (1 wait state with 15 ns SRAM at 66 MHz).
- The software simulator permits application tailoring since it provides clock exact behavior.
- In general, an algorithm that requires  $N$  clocks to execute and is repeated  $M$  times, requires ( $WS$  is a number of wait states):  
$$(N + N \times WS)M = N \times M(WS + 1) \text{ clocks.}$$
- In a cache environment, the same algorithm requires:  
$$N(WS + 1) + N(M - 1) = N(M + WS) \text{ clocks.}$$

## 8.8 Debugging Instruction Cache Operation

While the cache is enabled, full non-intrusive system debug capability in Debug mode includes being able to observe:

- What memory sectors are currently mapped into cache
- Which cache sectors are locked
- Which cache sector is the LRU
- When cache hits occur

Debug mode allows you to read the Tag register contents, lock bits, LRU bits, and hit-status serially from the OnCE module via the JTAG port. You can also read the valid bits of specific cache locations. To check whether an address with MSBs in a Tag register is in the cache, send the opcode of a MOVEM from this address. Bit 5 of the OnCE Status and Control register (OSCR) indicates the value of the valid bit. See **Chapter 7, *Debugging Support*** for more information.

**Note:** Each read of the cache status via the OnCE module should occur only when the device is in the Debug mode and should access all nine registers, so that reads start with tag #0 every time.